Analysis and measurement of software flexibility based on flexible points

Limin Shen, Shangping Ren

Abstract
Software flexibility is one of the properties that indicate if the software is easy to change. Flexible software can easily adapt to user requirement and/or environment changes during the software development period or after the software is deployed. In this paper, we introduce two new concepts, flexible change and FleXible Point (FXP). Associated with the flexible point, a set of flexibility indexes, i.e. flexible degree, flexible force, flexible distance, flexible capacity and their calculation formulas are defined. We further present a schema that utilizes these new concepts and indexes to quantitatively measure software flexibility. The software flexibility related properties. They are software adaptability, brittleness, rigidity, elasticity and plasticity, are also discussed and analysed. The flexible points are categorized into five different types: potential FXP, available FXP, used FXP, current FXP, and required FXP, and four different levels: Self-Adaptive FXP, Low-level User FXP, High-level User FXP, and Developer-Level User FXP. The relations and differences between flexible points and their impact on the software development, maintenance and quality are discussed. Measuring process of software flexibility based on flexible point is represented. A simple case study is carried out to illustrate the analysis and measurement of software flexibility. The metric in the paper can be used to guide software developers to determine and improve the flexibility of their software applications and to compare their flexibilities among different software systems or different version.

1. Introduction
Flexibility is often used in software engineering as one of the key properties. Many software products and technologies are labelled with “flexible”. However, what is software flexibility and how to measure it are yet to be clearly defined. Although intuitively software flexibility is understood as the ability to respond to changes, and is occasionally used for evaluating the quality of software, the formal definition of software flexibility and its properties are poorly understood. Such situation may be due to the following reasons:
• The “flexibility” is a word with rich meanings. It is sometimes used interchangeably with other terms, such as adaptability, plasticity, elasticity, agility and versatility, etc.
• Software flexibility, similar to material flexibility, is one of software’s internal properties. Though, how easy the software is to use, how long it takes to make changes on existing software, etc., to some degree reflect software flexibility properties, such external behaviours do not reveal the essence of software flexibility.
• To the author’s knowledge, there are no well-defined and unified software flexibility measurement criteria.

Since early 1980’s, the concept of “flexibility” has drawn attention in manufacture industry and business management [1][2][3]. In Flexible Manufacturing Systems (FMS), the flexibility denotes the ability to respond to changes and the ability to inflict such changes on other entities in the system. In software engineering, software flexibility concept had appeared in 1979 [4], but most of the literature came almost ten years later (in 1990’s) and the research has been primarily focused on the qualitative aspects of software flexibility [4][5][6][7][8].
Pamas (1979) thought flexible software is one that can be easily changed, extended, contracted, or else in order to be used in a variety of ways [4]. Keith Bennett (1999, 2000,2004) presented a list of features of a flexible software, suggesting that flexible software should be necessary and sufficient, personalized, adaptable and self-adaptive, distributed, in small units and transparent [5][6][7]. Nelson, et al. (1997) defines technology flexibility in the context of workflow management software. They proposed a measurement framework for technology flexibility that includes such factors as modularity, change dimension, rate of response, expertise and coordination of actions in the process flexibility dimension [8]. Zhao (1998) proposed two related software flexibility concepts in research on workflow management: system adaptability and system versatility. System adaptability is the capability to modify the system to cope with major changes in business processes with little or no interruption to business operations. System versatility (or system robustness) is the capability of the system to allow flexible procedures to deal with exceptions in processes and procedures [9]; Lassing (1999) made qualitative analysis of software flexibility on a management software based on given scenarios [10]; Zeng and Zhao (2002) presented “Achieving Software Flexibility via Intelligent Workflow Techniques” [11]. Keith Bennett (2004) explained software flexibility in such way that software is flexible – it is easy to change without compromising dependability (in broad terms, the cost of making a change is proportional to the size of the change, not the system) [7]. Fenton [3] proposes a measurement is carried out on software architecture, but the architecture generally is invisible to client.

Software is easy to change because it is “soft”, just a few keystrokes, but in practice it is difficult to change due to its intrinsic complexity and invisibility. It is often not clear when software should be changed, and when it should be discarded and replaced, what properties should remain unchanged. Up to now, software engineers have not been very successful in understanding how to do this, and the adaptation of software to changes has not been solved completely, yet the demands to change software in internet time are increasing. At present, a grand challenge facing software is coming from random changes of user requirement and environment. Only can software deal with these changes, it can survive for a long time because software is a logical entity, unable to be wore out or used up as hardware does, and software could live forever without changes of requirement and environment. Software flexibility is the property of software that software can change or be changed easily to satisfy the demand of user requirement and environment changes, and therefore software flexibility can be utilized to tackle the challenge. This paper attempts to develop a metric and a measurement method for software flexibility on functionality that can be used to:

• Compare different software flexibilities.
• Evaluate the capacity of software flexibility.
• Guide software developers and software users to carry out software evolution and further improve software flexibility.
• Prepare different level of software manipulators to apply software flexibility in order to increase software operation efficiency and bring software flexibility into play.

The rest of the paper is organized as follows: Section 2 develops the concept behind our metric for software flexibility measurement, and analyse the connotation of flexibility, adaptation, software change and flexible change. Section 3 proposes concepts and metrics of software flexibility measurement such as flexible degree, flexible force, flexible distance and flexible capacity, and their calculation formulas. Section 4 discusses the analysis based on flexible point types and levels and represents measuring process of software flexibility.
Section 5 further gives a case study to illustrate the analysis and measurement of software flexibility. Finally, section 6 draws conclusions and discusses future work.

2. Concept analysis

2.1. Flexibility and flexible change

Before the flexibility metric can be devised, certain concept analysis has to be made for flexibility and its related concepts. The thesaurus [12] gives the following interpretations for the two words “flexible” and “flexibility”: Capable of being bent, turned, bowed, flexed or twisted repeatedly without breaking, injury or damage; capable of being adapted; responsive to change; adaptable.

It is obvious that the original meaning of flexibility ought to be related to the properties of materials that are capable of being bent, turned, bowed, flexed or twisted repeatedly without being broken, or damaged. Accordingly, software flexibility is defined as the property of software that indicates if the software is easy to be changed or change repeatedly, inversely and harmoniously, i.e., software is capable of being bent, turned, bowed, flexed or twisted repeatedly without break, injury or damage. We name such changes as flexible changes. Flexible changes have the characteristics as follows:

- Change is controllable, i.e. change is limited to some range;
- Change is repeatable, i.e. change can be carried out repeatedly at almost the same ease;
- Change is inversed, i.e. change can be restored at almost the same ease;
- Change is harmonious, i.e. change is consistent and the change at one point does not lead to failure at other points, and that is easy to change without compromising dependability;
- Change is easy, i.e. change is accomplished with little time and low cost.

2.2. Flexibility and adaptability

Difference between flexibility and adaptability is subtle, sometimes even confusing. The adaptability refers to the extent to which a software system adapts to external changes or the ability to fit changed circumstances. The adaptability can be achieved through changing circumstances, software itself or other approach. It addresses that software accepts or endures external change in fast and easy way, but with less change. The adaptability exists with purposes and objectives, but the flexibility may exist with or without purposes and objectives. The flexibility addresses that software is able to achieve itself internal change easily. Therefore software flexibility can be exploited to adapt to external change, while adaptation mechanisms in software can make software change easily and enhance software flexibility.

A software that can be used in a variety of situations and can performs multiple missions simultaneously without change or modification is considered “adaptable” and “universal” not flexible according to the definition of flexibility provided above. Software flexibility is desirable in high uncertainty environment where software boundaries are subject to constant urgent change but not fixed. For example, emergent organisations are in a state of continual process change, such as the new e-businesses or traditional companies who continually need to change themselves to maintain competitive advantages. In order to adapt to such change, the software in the organisation often needs to change urgently, but the software without flexibility is often difficult to change and has high change cost. However, flexibility is a double-edged sword, risk but effective, for building flexibility in software needs extra effort and cost. Hence, it is important to measure and estimate software flexibility. Flexibility is one of the key indexes used for evaluating software ability of change on demand.
2.3. Analysis of software change

Besides flexibility there are other properties of software change as follows.

Software brittleness is a software property that indicates software is easy to be out of control or breakdown when an inappropriate manipulation, environment change, user requirement change or code change occurs.

Software rigidity is a software property that indicates software has a rigid specification and fixed boundaries, and resists changes. When an inappropriate manipulation is carried out, the software keep its state unchanged; when an appropriate manipulation is carried out, software converts user operation into the functionality execution. To rigid software, its code and data are difficult to be changed but its execution is reliable and stable.

Software elasticity is a software property that indicates software change is temporary, and the software is able to restores original form and state after it runs. The utilized functions are in proportion to the manipulation ability; data, code and configuration can be dynamically added into software, linked, bound or reconfigured, but these codes, data and configuration are merely used in an execution, and static composition keeps unchanged after an execution.

Software plasticity is a software property that indicates the partial software changes can be kept down in software, i.e. changed data, configuration and codes can be saved in software and they can be used at next execution.

Software animality is a software property that indicates software change does not completely depend on external manipulation, and is supported by software internal force, i.e., software can adjust, add, link, bind its data, code or configuration under external stimulation.

Brittleness, rigidity, elasticity, plasticity and animality reflect different feature of software changes. The brittleness, rigidity and elasticity have no memory about changes and changes do not impact its next execution; the plasticity and animality have memory about changes and changes impact its next execution. Elastic, plastic and animalism change can be flexible change. It is obvious that the points where change is not needed need rigidity and the points where change is needed need flexibility.

3. Measurement of software flexibility

3.1. Time, cost and software flexibility

In intuition, software flexibility has three dimensions: changeable range, time and cost. Software system is more flexible than another if it can handle a wider range of configurations, accommodate change in a shorter amount of time, or make the transition at lower cost. Time and cost are inversely related for cost may be reduced by more time for change, and time may be shortened at extra cost.

Time and cost are measurable, but they are relative value, not able to be used for definition, in that they cannot reflect the essence of flexibility. Due to its multidimensional nature, it was difficult to find a single value to measure flexibility and to compare flexibility among different software in the past.

In our view, as software is written by programming languages, hence, it has static composition and dynamic composition. The software static composition is its static form composed of static codes, frameworks, methods, data and other static information; while the software dynamic composition is its dynamic form, i.e., its runtime behaviours, composed of dynamic processes, threads, control flows, data flows, user’s interaction and other dynamic factors. Intuitively, software flexibility is one of software properties that indicate how easy the software can make the static composition changes and the dynamic composition changes. However, it is not easy to measure and evaluate ‘easy’.
3.2. Force and software flexibility

To study software flexibility, the “force” concept is introduced into software flexibility evaluation. It is the force that causes software change. The force $F$ consists of external force $F_e$ and internal force $F_i$, i.e., $F = F_e + F_i$. The internal force $F_i$ is given by the software itself. For example, software systems may provide adaptive interfaces that allow users to gain capacity for configuration or personalisation. The external force $F_e$ is given by the environment that the software interacts with, such as a user who manipulates software, a maintainer or a developer who modifies the software, and an application which controls the input stream, etc. From strength point of view, software flexibility is the ability that the software reacts to the force $F$. When software is changed, the less $F_e$ is needed, the easier change is, and the more flexible software is. Hence, the flexibility value ought to be determined by the outside force and the maximum change scope driven by it.

3.3. Measurement definition

Quantitative concepts and metric are introduced into software flexibility evaluation as follows:

- **Flexible Point (FXP) $i$**: a point or a location in software which can cause flexible changes to occur, upon which the external force $F_e$ may apply. $F_e$ causes software to changes through the flexible point. Small external force $F_e$ at a FXP may cause a large scale of changes in software. When $F_e = 0$, it indicates that the software changes are completely driven by internal force $F_i$.

- **Flexible Force $f_i$**: minimum external force $F_e$ applied to FXP $i$ that may cause software to change. $f_i$ indicates the easiness or difficulty to make software change. The larger $f_i$ is, the harder the software makes changes through FXP $i$.

- **Flexible Distance $S_i$**: maximum range or size of the software change caused by $f_i$ through flexible point $i$.

- **Flexible Degree $K_i$**: $K_i = S_i / (1 + f_i)$, a measure for software flexibility at FXP $i$.

- **Flexible Capacity $C$**: $C = \sum_{i=1}^{N} K_i$, a measure of entire or partial software flexibility.

Based on definitions above, a manipulator can utilize the flexibility at $i$ only if $f_e \geq f_i$.

4. Analysis and measurement process

4.1. Analysis based on flexible point type

Flexible points are categorized into five different types as follows:

1. Potential FXP (PFXP): all flexible points that are possibly used under an appropriate environmental setting. The capacity of PFXP, $C_{\text{PFXP}} = \sum_{i=1}^{N} K_i$.

2. Available FXP (AFXP): flexible points at which users have ability to manipulate. The capacity of AFXP is $C_{\text{AFXP}} = \sum_{i=1}^{N} K_i \mid f_i \leq F_e$.

3. Used FXP (UFXP): flexible points that users have manipulated and actually used. The capacity of UFXP is $C_{\text{UFXP}} = \sum_{i=1}^{N} K(i) \mid f_i \leq F_e$, and $i$ is a used flexible point. UFXP means that
users have used some flexible points, and the software has changed and some flexible mechanisms have been active, and flexibility effects have performed in software behaviours.

4 Current FXP (CFXP): flexible points that a current user is able to manipulate. The flexible capacity of CFXP is

\[ C = \sum_{i=1}^{N} K(i) \quad |f| \leq F_{\text{current user}} \]

5 Required FXP (RFXP): flexible points that are not in software and need to be added into software.

![Figure1: Using FXP to Adjust Software](image-url)

Obviously, \( PFXP \supseteq AFXP \supseteq UFXP \supseteq CFXP \). When a user puts forward a new change requirement to software, the user can use software’s FXP to adjust software functions. Figure1 shows adjustment steps. Inappropriate amounts of flexibility and unsuitable manipulators are indicated by significant discrepancies among the number of PFXP, AFXP, UFXP and RFXP. The problems can be reflected from availability and suitability.

Rate of FXP’s availability \( (RA) = \) number of AFXP/ number of PFXP. If RA is lower, that means that the manipulators are not unsuitable and manipulators should be adjusted.

Rate of FXP’s suitability \( (RS) = \) number of UFXP/ number of PFXP. If RS is lower, it means that some of software FXP are not appropriate, or many new FXP are required, or manipulators have no capability to operate them, or change requirements are few. If RS is lower and the number of RFXP is greater than the number of PFXP, designed FXP is not effective, and there are some serious problems in design of FXP.
4.2. Analysis based on flexible point level

Software manipulators can be general users, maintainers or developers, but their ability to manipulate software is different. The manipulators are divided into three levels: Low-level User (LU), High-level User (HU) and Developer-level User (DU). Their $F_e$ is $F_{LU}$, $F_{HU}$ and $F_{DU}$ separately. It is obvious, $F_{LU} \leq F_{HU} \leq F_{DU}$. Whether a user can utilize a FXP, it is determined by fact whether the manipulator’s $f_i$ is bigger than the FXP’s $f_i$, i.e. $F_i \geq f_i$. Based on the value range of $f_i$ and its relations with $F_e$, FXP can be distinguished as four different levels as follows.

1. Self-Adaptive Flexible Point (SAFP): flexible points with $f_i = 0$. They are oriented to all users, and able to detect the requirement and environment change at runtime and carry out self-configuring, self-optimising, self-protecting and self-healing actions. Its actions are transparent to users and completely driven by $F_{i\square}$. However, the SAFP effects are visible to users. $C_{SAFP} = \sum_{i=1}^{N} K(i) \mid f_i = 0$.

2. Low-level User Flexible Point (LUFP): flexible points with $0 < f_i \leq F_{LU}$. They are oriented to LU. $C_{LUFP} = \sum_{i=1}^{N} K(i) \mid 0 < f_i \leq F_{LU}$. LU is software routine user who only has basic knowledge about computers and business. By means of LUFXP, original software does not need developer modification, and LU can apply interface and operation provided by the software itself to adjust software functions such as setting parameters, defining a query etc.

3. High-level User Flexible Point (HUFP): flexible points with $F_{LU} < f_i \leq F_{HU}$. They are oriented to HU. $C_{HUFP} = \sum_{i=1}^{N} K(i) \mid F_{LU} < f_i \leq F_{HU}$. HU is software high-level user who has broad and in-depth knowledge about computers and the application domain. For instance, access control, security and manipulation ability, these key and complicated adjustments can only be made by HU. For example, by HUF, HU can adjust initial running environment, user interface, business rules, workflows and calculation formulas etc.

4. Developer-Level User Flexible Point (DUFP): flexible points with $F_{HU} < f_i \leq F_{DU}$. They are oriented to DU. $C_{DUFP} = \sum_{i=1}^{N} K(i) \mid F_{HU} < f_i \leq F_{DU}$. DU is the user with the highest $F_e$. DU has strong ability, experience and knowledge of business, system administration and software development. When LUFXP and HUFXP adjustment is not able to satisfy new user’s requirement, DU can utilize HUFXP to make deep adjustment to software by means of interface or platform such as adding codes or reconfiguration.

Related factors of FXP have relation as follow:

1. Implementation Difficulty (ID): $ID_{SAFP} > ID_{LUFP} > ID_{HUFP} > ID_{DUFP}$
2. Manipulation Easiness (ME): $ME_{SAFP} > ME_{LUFP} > ME_{HUFP} > ME_{DUFP}$
3. Manipulator Cost (MC): $MC_{SAFP} < MC_{LUFP} < MC_{HUFP} < MC_{DUFP}$

Because of $F_{LU} \leq F_{HU} \leq F_{DU}$, FXPs that LU can directly utilize are $ASFXP \cup LUFXP$, but LUFXP are suitable for LU. FXPs that HU directly utilizes are $ASFXP \cup LUFXP \cup HUFXP$, but HUFXP are suitable for HU. FXPs that DU directly utilize are $ASFXP \cup LUFXP \cup HUFXP \cup DUFXP$, but DUFXP are suitable for DU. It is obvious that the less the FXP’s $f_i$ is, the easier FXP is to be used. The bigger the user’s $F_e$ is, the more FXPs he can utilize. DU’s $F_e$ is the strongest, so he can utilize all FXPs.
Based on analysis above, there are approaches for users to increase the availability and the utilization of flexibility: one is to increase user’s ability to manipulate software; the other is to decrease \( f_i \) by improving software. If user’s \( F_i \) could reach the level of developers, he would utilize all FXP; if all FXP’s \( f_i \) could reach 0, all FXP could be utilized by any user.

4.3. Measuring process of software flexibility

4.3.1. Steps to measure software flexibility

The objective of this section is to introduce the steps necessary to measure and analyze software flexibility based on flexible point and metric proposed above. The steps is as follows:

- Identify flexible points.
- Analyse and calculate the flexible distance of every flexible point.
- Determine flexible point level and its flexible force value.
- Calculate the flexible degree of every flexible point.
- Calculate flexible capacity for different analysis.

4.3.2. Identification of flexible points

For flexible point is a location in software that can cause flexible changes in software, the flexible point can be a function, function control points, a reconfiguration, a segment of codes or a variant point etc. Sometimes flexible point is implicit, and it is necessary to make analysis according to the characteristics of flexible point proposed in section 2. Some general flexible points are shown as Table 1. Activities and operations at a flexible point can impact transaction function point components (EI, EO, EQ) and data function point components (ILF, ELF), and can cause them to complete flexible changes. By analysing impact scope of the flexible point, all the function point components changed can be found. It is the preparation needed by next step calculation of flexible distance.

<table>
<thead>
<tr>
<th>No</th>
<th>Change requirement</th>
<th>No</th>
<th>Change requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Adjust the value range of data element</td>
<td>6</td>
<td>Add /delete business rules</td>
</tr>
<tr>
<td>2</td>
<td>Add/delete items in selection</td>
<td>7</td>
<td>Add calculation formulas</td>
</tr>
<tr>
<td>3</td>
<td>Add/delete data element</td>
<td>8</td>
<td>Adjust screen layout</td>
</tr>
<tr>
<td>4</td>
<td>Change data element type</td>
<td>9</td>
<td>Change external input interface</td>
</tr>
<tr>
<td>5</td>
<td>Modify calculation formulas</td>
<td>10</td>
<td>Change external output interface</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.3. Calculation of flexible distance

Flexible distance is the range or size of the software change caused by a flexible point. However, it has been a difficult task to measure software change. We use a function point count as a unit of measure for software change. Because function points are a measure that represents the functional size of application software, it is more suitable and accurate taking function point counts to express software change size than using other unit. There are two main advantages of using function point as the unit of measure. One is that it is easier to locate, identify and determine changes because function point method classifies software function into the five components of function point [13]: External Inputs (EI), External Outputs (EO), External Inquiries (EI), Internal Logical Files (ILF),
External Interface Files (EIF); the other is that we can directly use the function point analysis approach given by International Function Point User Group (IFPUG) to calculate function point counts of changed function point components. The function point analysis process is following [14]:

- Determine type of function point count.
- Determine the application boundary.
- Identify and rate transactional function types to determine their contribution to the Unadjusted Function Point (UFP) count.
- Identify and rate data function types to determine their contribution to the UFP count.
- Determine the value of adjustment factor (VAF).
- Calculate the adjusted function point count.

Due to only concerning about the change size, we take UFP counts as flexible distance. Hence, when calculating flexible distance, we only need to carry out step 1 to 4.

### 4.3.4. Determination of flexible force value

Flexible force is minimum external force applied to a flexible point that can cause software to change. The value of flexible force is determined by flexible point level. Table 2 shows flexible point level and defines corresponding value of flexible force. At present, these values are defined based only on experiences, expert evaluation and intuition. How to evaluate these values is not within the scope of this paper.

<table>
<thead>
<tr>
<th>Flexible point level</th>
<th>Flexible force value</th>
<th>Manipulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAFXP</td>
<td>0</td>
<td>Not need user’s manipulation</td>
</tr>
<tr>
<td>LUFXP</td>
<td>10</td>
<td>Simple function manipulation</td>
</tr>
<tr>
<td>HUFXP</td>
<td>20</td>
<td>Complex function and business manipulation</td>
</tr>
<tr>
<td>DUFXP1</td>
<td>30</td>
<td>Low technical manipulation</td>
</tr>
<tr>
<td>DUFXP2</td>
<td>40</td>
<td>Average technical manipulation</td>
</tr>
<tr>
<td>DUFXP3</td>
<td>50</td>
<td>High technical manipulation</td>
</tr>
</tbody>
</table>

### 4.3.5. Calculation of flexible degree

Once flexible force $f_i$ and flexible distance $S_i$ are gained, the flexible degree of FXP $i$ can be calculated by following formula $K_i = S_i / (1 + f_i)$.

### 4.3.6. Calculation and analysis of flexible capacity

After every FXP’s level and flexible degree are determined, it is time to calculate flexible capacity of different type and level for different analysis. Section 5 will give a case study to demonstrate the calculation and analysis.

### 5. Case study

Table 3 gives a simple case study to demonstrate the calculation of flexible degree and flexible capacity. A Wage Calculation Software (WCS) has the elementary functions: data input, data print, inquiry, data import and data export. Meanwhile, WCS provides a number of flexible point functions to change software functionalities and behaviours as follows.

- Adjust the wage item width.
- Add/delete wage items.
• Modify calculation formulas.
• Adjust screen layout.
• Add calculation formulas.
• Adjust wage print table.

We propose three implementation schemes for flexible points. The level of flexible point in every scheme may be different. We can make comparison on flexibility of different schemes. On table 3, if \( f_i = \infty \), it means the flexible point does not exist. In scheme 1, we design FXP2, to add/delete wage items as HUFXP, which can make HU add or delete information items such as address, e-mail, birth date, etc and calculable items such as traffic allowance, worked hours, hour payment, etc without modifying code. We can see a relatively simple screen manipulation exposed to the end user at the flexible point, but after users add or delete wage items, WCS is able to automatically adjust and change data input interface, data output interface, internal data file structure, etc under the flexible point, i.e., function point components EI, EO, EQ, ILF and EIF are impacted and changed. This is a complex and robust process. If FXP 2 is designed as DUFXP, it needs developer’s intervention to satisfy requirement to add or delete wage items, and the flexibility at the flexible point would decrease.

We assume that prepared manipulators for WCS are LU and HU. Table 3 shows flexibility and its available rate of every scheme. Scheme 1 not only gains the highest flexibility, but also has the highest rate of availability because prepared manipulators accord with required manipulators. Inversely, scheme 3 not only gains the lowest flexibility, but also its availability rate is 0 because its required manipulator is DU but prepared manipulators are HU. The data on table 3 can be used as quantitative facts to guide software developers, organisations and users to carry out improvement of software flexibility.

### Table 3: Flexibility calculation of WCS

<table>
<thead>
<tr>
<th>No</th>
<th>Flexible points</th>
<th>Changed Component</th>
<th>( S_i )</th>
<th>Scheme 1</th>
<th>Scheme 2</th>
<th>Scheme 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(UFP)</td>
<td>( f_i )</td>
<td>( K_i )</td>
<td>( f_i )</td>
<td>( K_i )</td>
</tr>
<tr>
<td>1</td>
<td>Adjust the wage item width</td>
<td>ELEO, EQ, ,ILF, EIF</td>
<td>56</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Add/delete wage item</td>
<td>ELEO, EQ, ILF, EIF</td>
<td>178</td>
<td>20</td>
<td>8.48</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>Modify calculation formulas</td>
<td>EO, ILF</td>
<td>29</td>
<td>10</td>
<td>2.64</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>Adjust screen layout</td>
<td>EI</td>
<td>6</td>
<td>0</td>
<td>6</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>Add calculation formulas</td>
<td>EO, ILF</td>
<td>30</td>
<td>20</td>
<td>1.42</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Adjust wage print table</td>
<td>EO, EQ</td>
<td>12</td>
<td>10</td>
<td>1.09</td>
<td>20</td>
</tr>
</tbody>
</table>

| Required manipulators | LU, HU | HU, DU | DU |
| Prepared manipulators | LU, HU | LU, HU | LU, HU |
| Potential flexible capacity | 22.3 | 6.53 | 3.33 |
| Available flexible capacity | 22.3 | 4.90 | 0.00 |
| Available rate of flexibility | 100% | 75% | 0% |
| SA flexible capacity | 0 | 0 | 0 |
| LU flexible capacity | 3.73 | 0 | 0 |
| HU flexible capacity | 12.57 | 4.91 | 0 |
| DU flexible capacity | 0 | 1.62 | 3.33 |
6. Conclusion

Many of the benefits of flexibility are intangible, and they are difficult to quantify. However, to evaluate and improve software flexibility, on the other hand, require more quantitative approaches. On the basis of concept of flexible point developed in this paper, the connotation of software flexibility is explored. Elementary metric for software flexibility such as flexible degree, flexible force, flexible distance and flexible capacity are proposed. We have shown a case how these metrics are applied.

This is our initial investigation on measurement of software flexibility. There is still a lot of work yet to be done. One of the tasks is to provide a formal proof for the evaluation of flexible force. We need further improve function point method to suit the measurement of flexible distance. Despite yet to be finished tasks, our work provides a new way to understand and answer questions of the following:

- What is the essence of software flexibility?
- Why, where, when, and how much software is software flexibility needed?
- What are the trade-offs associated with designing flexible software?
- How to measure software flexibility and what are the penalties for such flexibility?

7. Reference